



Digital Signal Processing
Mini-Project Report

Submitted by

Name : Sarang Galada

Register Number : 210200001

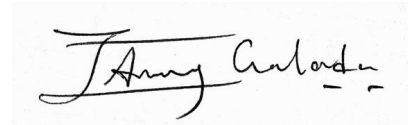
E-mail ID : sarang.g-25@scds.saiuniversity.edu.in

School of Study : Computing & Data Science

Year of Study : III

Declaration

I hereby declare that the work presented in this mini-project report is my own. All sources used in this work have been properly cited and referenced.

A handwritten signature in black ink, appearing to read 'Sarang Galada', is centered on a light gray rectangular background.

Sarang Galada

Date: 13/11/2023

DCT IMAGE COMPRESSION

1. Problem Statement

This project aims to develop a lossy image compression system utilising the Discrete Cosine Transform, built from the ground up.

2. Motivation

In an era characterised by the exponential proliferation of digital information, there is a vital need to devise strategies for efficient storage and transmission of data. The rise of social media platforms and the surge in image content have spurred considerable research efforts into image data compression techniques, aiming to enhance storage and transmission efficiency [1]. This project explores the Discrete Cosine Transform (DCT), one such image compression technique that is widely used and offers comparative advantages.

3. Methodology

The Discrete Cosine Transform (DCT) is a signal processing technique for converting a signal into elementary frequency components [2]. It is widely used in image/video compression, for example, in the JPEG/MPEG standards. It was originally derived from the Discrete Fourier Transform (DFT), but unlike DFT, provides very high *energy compaction* and doesn't involve the use of complex values. This makes it ideal for usage in lossy image compression. The analysis and synthesis equations for DCT are as follows [3]:

Analysis equation (Forward DCT)

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right],$$

In kernel matrix form: $F(x,y) = C1 * f(x,y) * C2^T$

Synthesis equation (Inverse DCT)

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u)\alpha(v) C(u,v) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right],$$

In kernel matrix form: $f(x,y) = C1^T * F(x,y) * C2$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u \neq 0. \end{cases}$$

4. Implementation

In this project, we develop from scratch a computer program written in MATLAB to achieve lossy image compression by utilising DCT along with quantization. The implementation comprises the following steps:

- 1) Accept any input image file and convert it to grayscale.
- 2) Apply DCT to get the transformed values.
- 3) Quantizing the values based on a user-defined quantization factor (Q), resulting in lossy compression of information.

Quantization formula: $q_img = Q * \text{round} (img / Q)$

- 4) Applying Inverse DCT to reconstruct the image in compressed form.

It may be noted here that this process includes the crucial steps of the JPEG compression algorithm, namely the use of DCT and quantization, but with the entropy encoding step omitted [4].

MATLAB Code: [5]-[7]

```
classdef dct_img_compressor_exported < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                matlab.ui.Figure
        CompressionFactorSlider  matlab.ui.control.Slider
        CompressionFactorSliderLabel  matlab.ui.control.Label
        CompressButton           matlab.ui.control.Button
        DownloadImageButton       matlab.ui.control.Button
        UploadImageButton         matlab.ui.control.Button
        UIAxes                   matlab.ui.control.UIAxes
        UIAxes2                  matlab.ui.control.UIAxes
        VariableTextLabel         matlab.ui.control.Label
    end
```

```

% Callbacks that handle component events
methods (Access = private)
    % Button pushed function: UploadImageButton
    function UploadImageButtonPushed(app, event)

        % Allow user to select an image
        [file, path] = uigetfile({'*.jpg;*.png;*.bmp', 'Image Files (*.jpg, *.png, *.bmp)'}, 'Select an Image');
        if isequal(file, 0)
            % User canceled the operation
            return;
        end
        % Load the selected image
        img = imread(fullfile(path, file));

        % If image is RGB, convert to grayscale
        if size(img, 3) == 3
            img = rgb2gray(img);
        end
        img = im2double(img);
        % Display the original image
        imshow(img, 'Parent', app.UIAxes);
    end
    % Value changed function: CompressionFactorSlider
    function CompressionFactorSliderValueChanged(app, event)
        value = app.CompressionFactorSlider.Value;
        setappdata(0, 'value', value);
    end

    % Button pushed function: DownloadImageButton
    function DownloadImageButtonPushed(app, event)

        % Allow user to save the compressed image
        [file, path] = uiputfile({'*.jpg', 'JPEG Image (*.jpg)'}, 'Save Compressed Image');
        if isequal(file, 0)
            % User canceled the operation
            return;
        end
        % Get the compressed image from the UIAxes
        compressed_img = getimage(app.UIAxes);
        % Save the compressed image
        imwrite(compressed_img, fullfile(path, file));
    end
end

```

```

% Button pushed function: CompressButton
function CompressButtonPushed(app, event)
    image = getimage(app.UIAxes);

    % If image is RGB, convert to grayscale
    if size(image, 3) == 3
        image = rgb2gray(image);
        image = im2double(image);
    end

    % Generate the first DCT kernel (C1)
    [M,N] = size(image);
    i = (0:M-1);
    j = (0:M-1);
    C0 = cos(pi*(2*i'+1)*j/(2*M));
    C1 = C0 * sqrt(diag([1, 2*ones(1,M-1)])/M);

    % Generate the second DCT kernel (C2)
    k = (0:N-1);
    l = (0:N-1);
    C0 = cos(pi*(2*k'+1)*l/(2*N));
    C2 = C0 * sqrt(diag([1, 2*ones(1,N-1)])/N);

    % Apply DCT
    img_dct = C1*image*C2';

    % Apply Quantization using quantization factor Q
    Q = getappdata(0, 'value');
    img_dct = Q*(round(img_dct/Q));

    % Apply Inverse DCT to the quantized image
    compressed_img = C1'*img_dct*C2;

    % Display the compressed image
    imshow(compressed_img, 'Parent', app.UIAxes2);

    % Calculate the compression ratio
    compression_ratio = numel(image)/nnz(img_dct);

    % Display the variable
    app.VariableTextLabel.Text = ['CR: ',
    num2str(compression_ratio)];
end
end

```

```

% Component initialization
methods (Access = private)
    % Create UIFigure and components
    function createComponents(app)
        % Create UIFigure and hide until all components are created
        app.UIFigure = uifigure('Visible', 'off');
        app.UIFigure.Position = [100 100 640 480];
        app.UIFigure.Name = 'DCT Image Compressor';

        % Create UIAxes2
        app.UIAxes2 = uiaxes(app.UIFigure);
        title(app.UIAxes2, 'Compressed Image')
        app.UIAxes2.XTick = [];
        app.UIAxes2.YTick = [];
        app.UIAxes2.Position = [350 161 283 267];

        % Create UIAxes
        app.UIAxes = uiaxes(app.UIFigure);
        title(app.UIAxes, 'Original Image')
        app.UIAxes.XTick = [];
        app.UIAxes.YTick = [];
        app.UIAxes.Position = [9 161 283 267];

        % Create UploadImageButton
        app.UploadImageButton = uibutton(app.UIFigure, 'push');
        app.UploadImageButton.ButtonPushedFcn = createCallbackFcn(app,
@UploadImageButtonPushed, true);
        app.UploadImageButton.BusyAction = 'cancel';
        app.UploadImageButton.Position = [97 104 100 22];
        app.UploadImageButton.Text = 'Upload Image';

        % Create DownloadImageButton
        app.DownloadImageButton = uibutton(app.UIFigure, 'push');
        app.DownloadImageButton.ButtonPushedFcn = createCallbackFcn(app,
@DownloadImageButtonPushed, true);
        app.DownloadImageButton.Position = [437 104 104 22];
        app.DownloadImageButton.Text = 'Download Image';

        % Create CompressButton
        app.CompressButton = uibutton(app.UIFigure, 'push');
        app.CompressButton.ButtonPushedFcn = createCallbackFcn(app,
@CompressButtonPushed, true);
        app.CompressButton.Position = [272 40 100 22];
        app.CompressButton.Text = 'Compress';

```

```

    % Create CompressionFactorSliderLabel
    app.CompressionFactorSliderLabel = uilabel(app.UIFigure);
    app.CompressionFactorSliderLabel.HorizontalAlignment = 'right';
    app.CompressionFactorSliderLabel.Position = [265 140 113 22];
    app.CompressionFactorSliderLabel.Text = 'Quantization Factor';

    % Create CompressionFactorSlider
    app.CompressionFactorSlider = uislider(app.UIFigure);
    app.CompressionFactorSlider.Limits = [0 1];
    app.CompressionFactorSlider.MajorTicks = [0 0.5 1];
    app.CompressionFactorSlider.ValueChangedFcn =
createCallbackFcn(app, @CompressionFactorSliderValueChanged, true);
    app.CompressionFactorSlider.Position = [285 120 74 3];
    app.CompressionFactorSlider.Value = 0.001;

    % Create VariableTextLabel
    app.VariableTextLabel = uilabel(app.UIFigure);
    app.VariableTextLabel.HorizontalAlignment = 'left';
    app.VariableTextLabel.Position = [437 40 200 22];
    app.VariableTextLabel.Text = 'Compression Ratio: ';

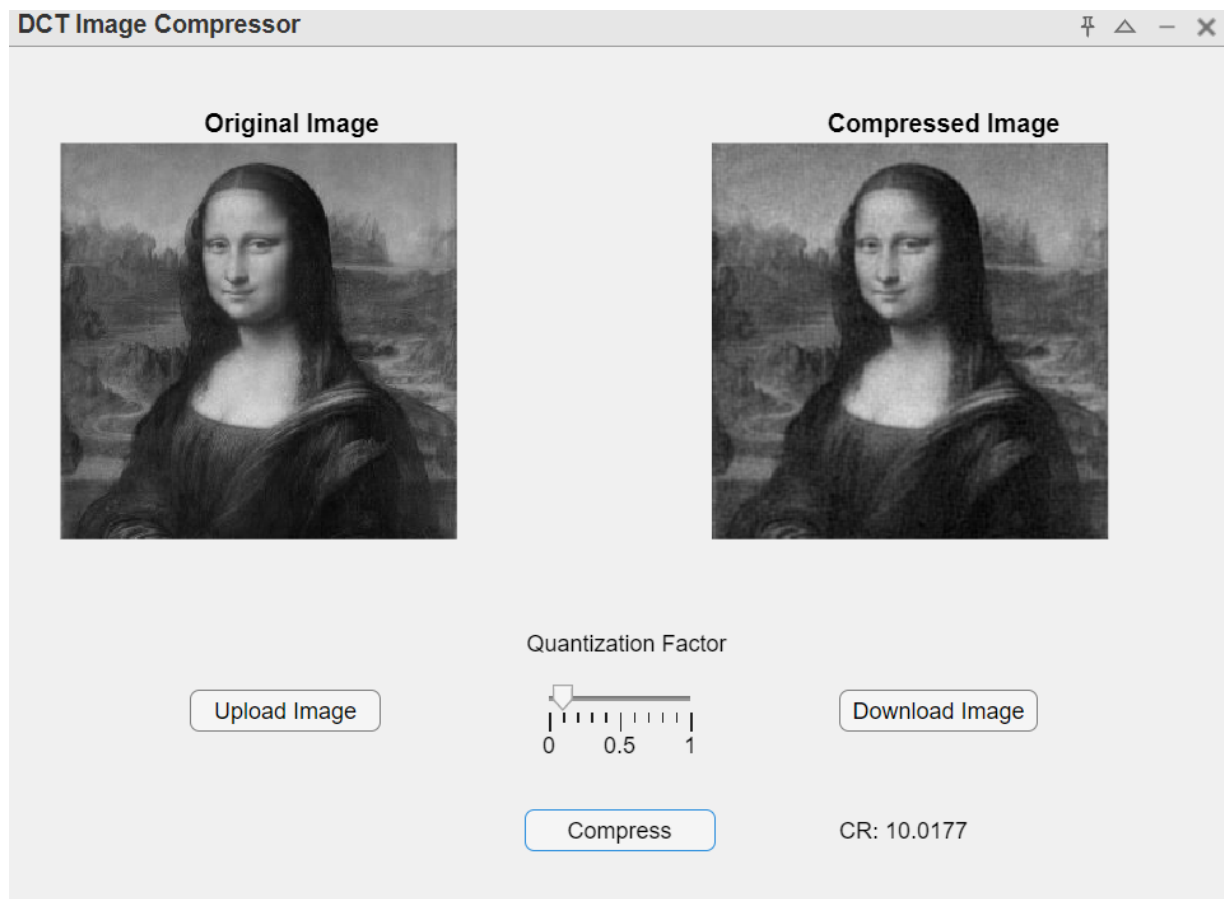
    % Show the figure after all components are created
    app.UIFigure.Visible = 'on';
end
end

% App creation and deletion
methods (Access = public)
    % Construct app
    function app = dct_img_compressor_exported
        % Create UIFigure and components
        createComponents(app)
        % Register the app with App Designer
        registerApp(app, app.UIFigure)
        if nargin == 0
            clear app
        end
    end
    % Code that executes before app deletion
    function delete(app)
        % Delete UIFigure when app is deleted
        delete(app.UIFigure)
    end
end
end
end

```

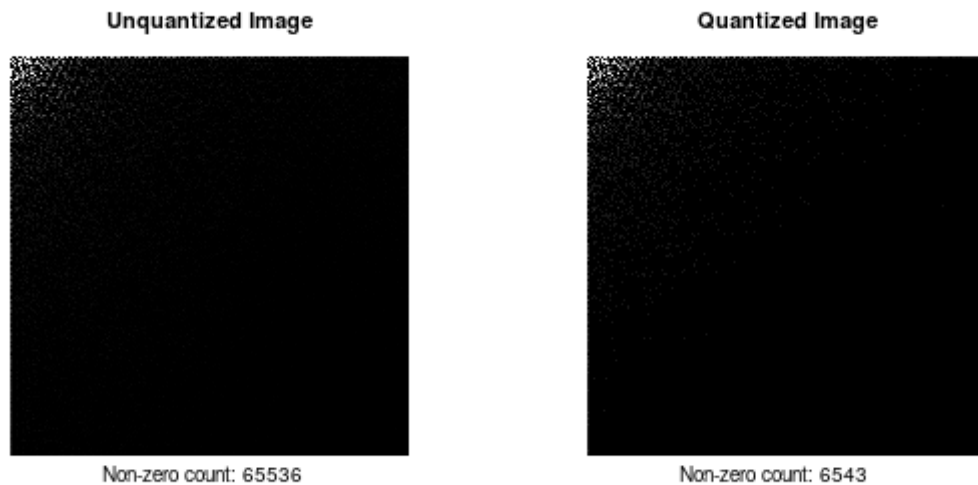

5. Simulation Results

As can be seen from the following results, the lossy image compression application works successfully.



By adjusting the Quantization Factor, we see that a Compression Ratio (CR = # elements in input image / # non-zero elements in quantized DCT image) of 10:1 results in minor perceptible loss in image quality. This tallies with the typical compression ratio achieved by the JPEG compression standard, which is usually 10:1 or 20:1 [8].

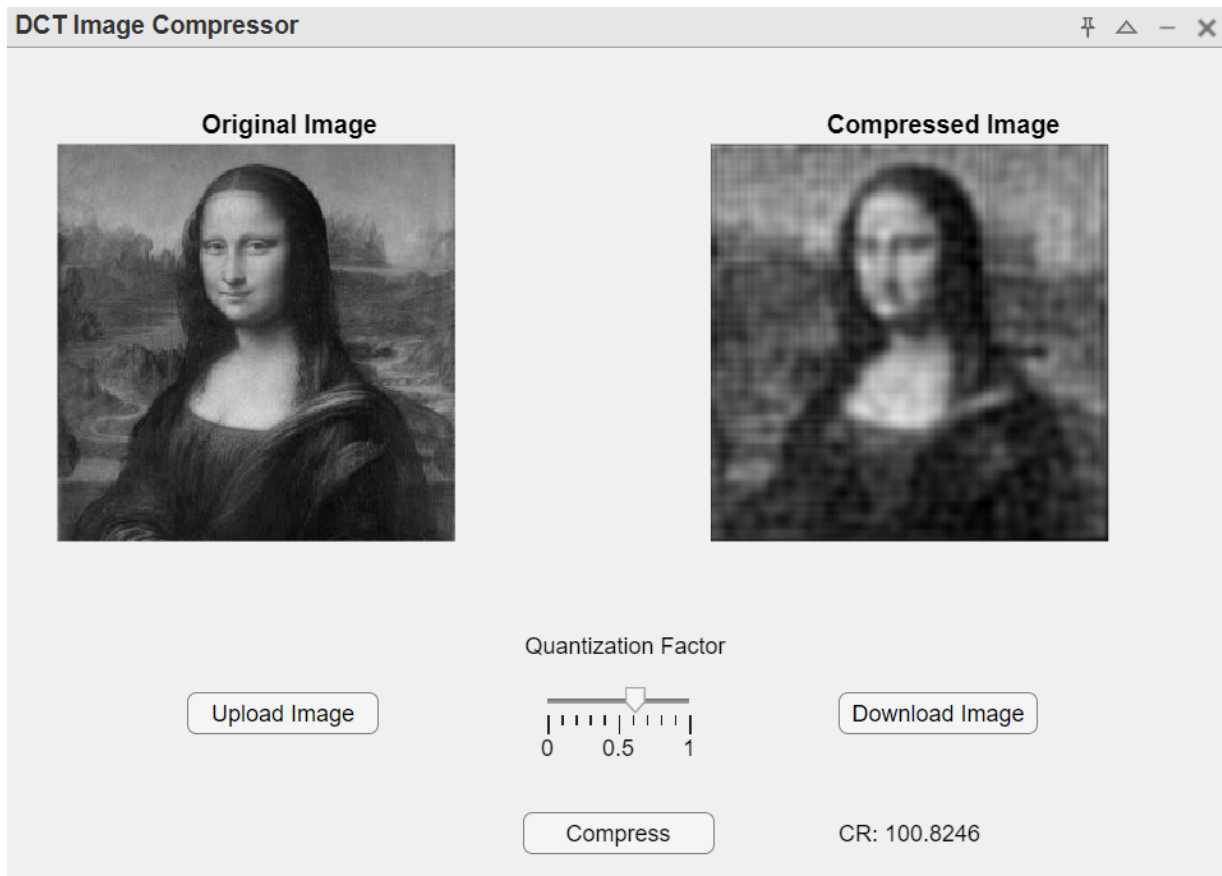
Analysing the intermediary images in the transformed domain for CR=10:1, before and after quantization, we get:



It can be observed that most of the transformed image is black, ie. contains less-significant values (close to zero or negative), and only pixels in the top-left corner contain significant values.

After quantization, the non-zero value count drops tenfold, since all less-significant values are quantized to zero and the remaining non-zero values are truncated to a lower precision.

Thus, when an entropy encoding algorithm such as Huffman encoding is applied, the storage space required is far lesser than that of the original image, allowing for both efficient storage and transmission.



Upon increasing the Compression Ratio, it is found that the point at which it becomes difficult to discern the image's features is around the 100:1 mark. The practical implication of this is that we can reduce our image to almost $1/100^{\text{th}}$ the original size and yet retrieve back from it an image that largely conveys the nature of the original image.

6. Conclusion

From this endeavour we can conclude that it is indeed possible and effective to use the Discrete Cosine Transform to compress images. The demonstrations highlight the immense energy compaction property of DCT and its practical efficacy in lossy image compression.

Furthermore, by building a basic MATLAB GUI application from the ground up (without using in-built functions), we were able to successfully compress images by applying DCT and quantization.

References

- [1] A.M. Raid et al, "Jpeg Image Compression Using Discrete Cosine Transform - A Survey." arXiv:1405.6147 [cs.MM], International Journal of Computer Science & Engineering Survey (IJCSES) Vol.5, No.2, April 2014. [Abstract]. Available: <https://arxiv.org/abs/1405.6147>. [Accessed November 14, 2023].
- [2] Watson, Andrew, "Image Compression Using the Discrete Cosine Transform." *Mathematica Journal*, 4(1), p. 81-88, August 1994. [Abstract]. Available: https://www.researchgate.net/publication/2800608_Image_Compression_Using_the_Discrete_Cosine_Transform. [Accessed November 14, 2023].
- [3] sandz24, "Discrete Cosine Transform (DCT) Formula," Daniweb.com, May 02, 2012. [Online]. Available: <https://www.daniweb.com/programming/software-development/threads/422269/discrete-cosine-transform-dct-formula>. [Accessed

November 19, 2023].

[4] Shawahna, Ahmad et al, "JPEG Image Compression using the Discrete Cosine Transform: An Overview, Applications, and Hardware Implementation." arXiv:1912.10789 [cs.MM], 1 Nov, 2019. Available: <https://arxiv.org/abs/1912.10789>. [Accessed November 16, 2023]

[5] Florian, "DCT with MATLAB," DSP Stack Exchange, Sep 30, 2020. [Online]. Available: <https://dsp.stackexchange.com/questions/70614/dct-with-matlab>. [Accessed November 12, 2023].

[6] Bharadi, Vinayak, "DCT Implementation in C#," CodeProject.com, November 16, 2009. [Online]. Available: <https://www.codeproject.com/Articles/43782/DCT-Implementation-in-C-DCT-of-Image>. [Accessed October 25, 2023].

[7] Eddins, Steve, "Revisiting dctdemo," Mathworks Blog, February 8, 2013. [Online]. Available: <https://blogs.mathworks.com/steve/2013/02/08/revisiting-dctdemo-part-1/>. [Accessed November 13, 2023].

[8] J.D. Paola, R.A. Schowengerdt, "The Effect of Lossy Image Compression on Image Classification." Research Institute for Advanced Computer Science, NASA Ames Research Center, August 16, 1995. Available: <https://ntrs.nasa.gov/citations/19960003362>. [Accessed November 17, 2023]